**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по Курсовой работе**

**по дисциплине «Программирование»**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 0324 |  | Жигалова Д.А. |
| Преподаватель |  | Глущенко А.Г |

Санкт-Петербург

2021

# **Цель работы**

Формирование общей программы по двум лабораторным работам с инфраструктурой переключения между заданиями (интерактивное меню).

# **Основные теоретические положения**

Структуры представляют собой группы связанных между собой, как правило, разнотипных переменных, объединенных в единый объект, в отличие от массива, все элементы которого однотипны. В языке C++ структура является видом класса и обладает всеми его свойствами.

Для определения структуры применяется ключевое слово struct, а сам формат определения выглядит следующим образом:

struct [имя\_типа] {

тип\_1 элемент\_1;

тип \_2 элемент\_2;

…

тип\_k элемент\_k;

} [ список\_описателей ];

Вся эта конструкция является инструкцией языка программирования, поэтому после нее всегда должен ставиться символ ‘;’.

struct Student

{

int age;

char name[10];

int group;

};

После определения структуры можно её использовать. Для начала можно определить объект структуры - по сути обычную переменную, которая будет представлять описанную ранее структуру.

struct Student Ivan;

Здесь определена переменная Ivan, которая представляет структуру Student.

При описании структуры память для размещения данных не выделяется. Работать с описанной структурой можно только после того, как будет определена переменная (переменные) этого типа данных, только при этом компилятор выделит необходимую память.

Еще один способ определения структуры представляет ключевое слово typedef:

typedef struct

{

int age;

char name[10];

} person;

В конце определения структуры после закрывающей фигурной скобки идет ее обозначение - в данном случае person. В дальнейшем можно использовать это обозначение для создания переменной структуры.

person tom = {23, "Tom"};

Еще один способ определить структуру представляет применение препроцессорной директивы #define:

#define PERSON struct {int age; char name[20];}

int main(void)

{

PERSON tom = {23, "Tom"};

return 0;

}

В данном случае директива define определяет константу PERSON, вместо которой при обработке исходного кода препроцессором будет вставляться код структуры struct {int age; char name[20];}

Каждая входящая в структуру переменная называется членом (полем, элементом) структуры и описывается типом данных и именем. Поля структуры могут быть любого типа данных. Их количество не лимитировано.

Структуры можно использовать в качестве параметров функций, как и обычные переменные. Для структур поддерживаются все три механизма передачи данных: по значению, через указатели и по ссылке.

Передачу структур в функции по значению необходимо использовать аккуратно:

void WriteStudent ( t\_Student S )

{

cout << "Фамилия: " << S.Fam << endl;

cout << "Имя: " << S.Name << endl;

cout << "Год рождения: " << S.Year << endl;

if ( S.Sex )

cout << "Пол: " << "М\n";

else

cout << "Пол: " << "Ж\n";

cout << "Средний балл: " << S.Grade << endl;

}

Вызов такой функции сопровождается дополнительным расходом памяти для создания локальной переменной S и дополнительными затратами времени на физическое копирование данных из аргумента в параметр S. Учитывая то, что объем структур может быть очень большим, эти дополнительные затраты вычислительных ресурсов могут быть чрезмерными.

Предпочтительно использование передачи структуры по указателю или ссылке:

void WriteStudent ( t\_Student \*S )

{

cout << "Фамилия: " << S -> Fam << endl;

cout << "Имя: " << S -> Name << endl;

cout << "Год рождения: " << S -> Year << endl;

if ( S -> Sex )

cout << "Пол: " << "М\n";

else

cout << "Пол: " << "Ж\n";

cout << "Средний балл: " << S -> Grade << endl;

}

Фактической передачи данных в функцию не осуществляется. Дополнительные затраты памяти для создания локальной переменной небольшие – это адрес памяти (4 байта, независимо от размера самой структуры). Вызов такой функции будет происходить быстрее, а расход памяти будет существенно меньше, чем при передаче данных по значению.

Передача по ссылке по эффективности эквивалентна передаче данных через указатель. Однако, поскольку при передаче данных по ссылке все адресные преобразования берет на себя компилятор, существенно упрощается программирование действий со структурами. При использовании ссылочных параметров структурных типов доступ к членам структуры осуществляется обычным способом – с помощью оператора «точка».

Недостатком этих способов является то, что случайные изменения значений полей структуры внутри функции отразятся на значении аргумента после окончания работы функции. Если необходимо предотвратить изменения переданных по адресу аргументов, можно при определении соответствующего параметра объявить его константой (использовать спецификатор const).

**Линейные структуры данных. Динамические массивы и двусвязные списки.**

Схема распределения памяти под программу показана на рис. 1.

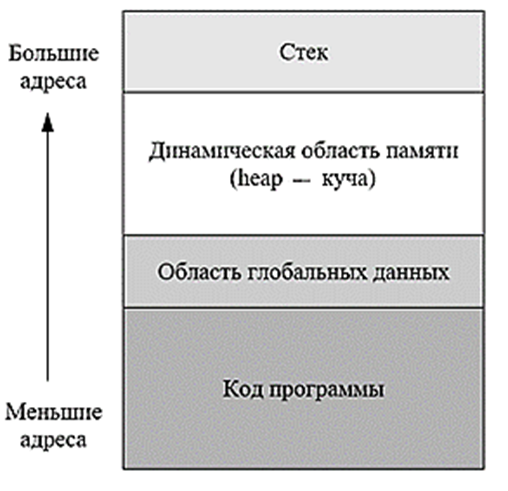


Рис. 1. Схема распределения памяти под программу

Область кода программы предназначена для хранения инструкций функций программы, обеспечивающих обработку данных.

Данные в программе представляются переменными и константами.

Для хранения глобальных данных предназначена область глобальных данных.

Стек программы используется при вызове функций для передачи параметров и хранения локальных данных.

Распределение памяти для хранения всех обычных переменных осуществляется компилятором, адреса и объемы соответствующих участков памяти (в области глобальных данных) жестко закреплены за этими переменными на все время работы программы и изменены быть не могут.

Однако во многих задачах невозможно заранее предсказать, сколько места (количество переменных, объемы массивов и т. д.) потребуется для решения задачи – это так называемые задачи с неопределенной размерностью.

Решить эту проблему можно лишь в том случае, если иметь механизм, позволяющий создавать новые объекты по мере возникновения необходимости в этих объектах или изменять объемы памяти, выделенные под эти объекты (например, объемы массивов).

Между областью глобальных данных и стеком располагается так называемая динамическая область памяти, которую и можно использовать в процессе работы программы для реализации механизма динамического управления памятью.

Для того чтобы создать в динамической области некоторый объект, необходима одна обычная переменная-указатель (не динамическая переменная). Сколько таких объектов понадобится для одновременной обработки, столько необходимо иметь обычных переменных-указателей. Таким образом, проблема задач неопределенной размерности созданием одиночных динамических объектов решена быть не может.

Решить эту проблему поможет возможность создавать в динамической области памяти массивы объектов с таким количеством элементов, которое необходимо в данный момент работы программы, т. е. создание динамических массивов. Действительно, для представления массива требуется всего одна переменная-указатель, а в самом массиве, на который ссылается этот указатель, может быть столько элементов, сколько требуется в данный момент времени.

Для создания одномерного динамического массива, элементами которого являются, например, действительные числа, используется следующий синтаксис инструкции new (стиль С++):

double \*Arr = new double [100];

Освободить динамическую область от этого массива можно с помощью инструкции delete:

delete [] Arr;

После этого занятый участок памяти будет возвращен в список свободной памяти и может быть повторно использован для размещения других динамических объектов.

Язык C++ поддерживает и «старый», заимствованный от языка C, стиль работы с динамической областью. Довольно часто бывает полезно использовать именно этот механизм управления динамической памятью.

В языке C отсутствуют инструкции new и delete. Вместо них для управления динамической памятью используются библиотечные функции:

// Блок прототипов функций

{

void \*malloc (size);

void \*calloc(num, size);

void free( void \*memblock);

void \*realloc( void \*memblock, size);

}

Функция malloc выделяет в динамической области size байт памяти и возвращает адрес этого участка в виде указателя (void \*).

Поскольку возвращаемый указатель не привязан ни к какому типу данных, при работе с ним потребуется явное приведение типов данных (см. пример далее).

Функция calloc выделяет в динамической области size \* num байт памяти и возвращает адрес этого участка в виде указателя (void \*).

Функция free освобождает участок динамической памяти по адресу memblock и возвращает его в список свободной памяти для повторного использования.

Функция realloc позволяет изменить размер (уменьшить или увеличить) ранее выделенной по адресу memblock памяти, установив новый размер выделенного участка равным size байт. При увеличении размера выделенного участка данные, которые хранились в старом участке, копируются в новый участок памяти. При уменьшении объема выделенного участка данные, которые хранились в нем, усекаются до нового размера. Функция возвращает указатель на область памяти нового размера.

Работа с одномерным динамическим массивом осуществляется так же, как и с обычным. При этом стиль использования динамических массивов С имеет весомое преимущество над С++, которое заключается в изменении размерности массива. Дело в том, что в C++ нет функций увеличения размерности. Увеличить размер массива можно, создав новый динамический массив нужной размерности, скопировав данные из старого массива в новый и освободив память от старого массива.

Одномерный однонаправленный список представляет собой совокупность отдельных элементов, каждый из которых содержит две части – информационную ("Data" ) и адресную ("Tail" ).

Информационная часть предназначена для хранения полезных данных и может иметь практически любой тип. Адресная часть каждого элемента содержит адрес следующего элемента списка. Схематическое изображение такого списка представлено на рис. 2.

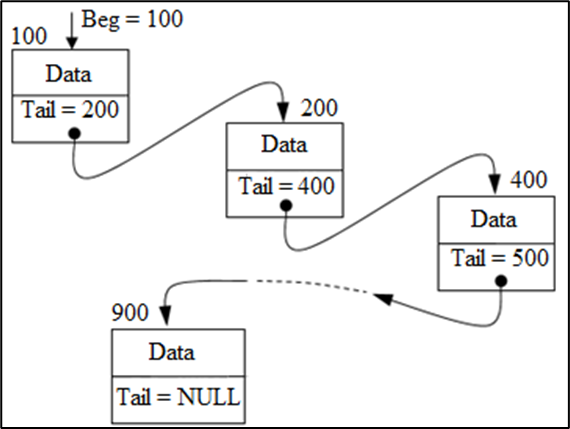


Рис. 2. Схематическое изображение односвязного списка

Для работы со списком достаточно знать только адрес его первого элемента ("Beg" ). Зная адрес первого элемента списка можно последовательно получить доступ к любому другому его элементу.

Поскольку каждый элемент списка должен иметь две части, логичнее всего представить его в виде следующей структуры:

struct list

{

int data;

list \*tail;

};

Типовыми операциями при работе со списками являются:

создание списка;

освобождение памяти от списка (удаление списка);

доступ к заданному элементу списка для манипуляций с его информационной частью;

добавление нового элемента к списку;

удаление элемента из списка;

перестановка элемента списка на новую позицию внутри списка.

Достоинством подобных структур является простота добавления, удаления и перестановки элементов списка, которые осуществляются путем манипуляций с адресными частями без перезаписи всего списка.

Одним из недостатков односвязных списков является то, что узел (элемент списка) имеет указатель только на следующий элемент. Вернуться из текущего элемента к предыдущему явным способом невозможно.

Каждый узел двусвязного (двунаправленного) линейного списка содержит два поля указателей – на следующий и на предыдущий узлы. Указатель на предыдущий узел корня списка содержит нулевое значение. Указатель последнего узла также содержит нулевое значение.

Поскольку каждый элемент списка должен иметь три части, логичнее всего представить его в виде следующей структуры:

struct list

{

int data;

list \*head;

list \*tail;

};

На рис. 3 показано схематическое представление двусвязного списка. Поле "Head" содержит адрес предыдущего элемента, поле "Tail " содержит адрес следующего элемента списка. Такая организация списка позволяет перемещаться по его элементам в двух направлениях.

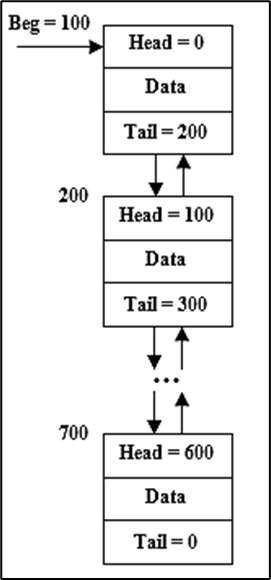


Рис. 3. Схематическое изображение двусвязного списка

Основные действия, производимые над узлами двусвязного линейного списка (ДЛС):

инициализация списка;

добавление узла в список;

удаление узла из списка;

удаление корня списка;

вывод элементов списка;

вывод элементов списка в обратном порядке;

взаимообмен двух узлов списка.

Порядок действия очень похож на односвязный линейный список, но необходимо учитывать, что в двусвязном списке имеется два указателя: на следующий и предыдущий элементы.

**Польские нотации. Стек и очередь**

Структуры представляют собой группы связанных между собой, как правило, разнотипных переменных, объединенных в единый объект, в отличие от массива, все элементы которого однотипны. В языке C++ структура является видом класса и обладает всеми его свойствами.

Стек – это частный случай однонаправленного списка, добавление элементов в который и выборка из которого выполняется с одного конца, называемого вершиной стека. Другие операции со стеком не определены. При выборке элемент исключается из стека. Говорят, что стек реализует принцип обслуживания LIFO (последним пришел – первым ушел).

Основные операции над стеками:

1. чтение верхнего элемента;
2. добавление нового элемента;
3. удаление существующего элемента.

Графически его удобно изобразить в виде вертикального списка (рис. 1), например, стопки книг, где для того чтобы воспользоваться одной из них и не нарушить установленный порядок, нужно поднять все книги, которые лежат выше нее, а положить книгу можно лишь поверх всех остальных.

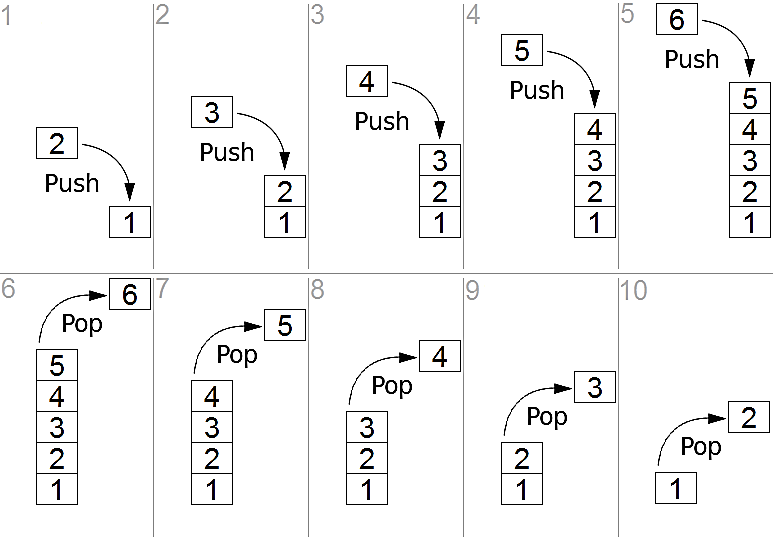


Рис. 1. Графическое представление стека

Стек чаще всего реализуется на основе обычных массивов, односвязных и двусвязных списков. В зависимости от конкретных условий выбирается одна из этих структур данных.

Очередь – частный случай однонаправленного списка, добавление элементов в который выполняется в один конец, а выборка – из другого конца. Другие операции с очередью не определены. При выборке элемент исключается из очереди. Говорят, что очередь реализует принцип обслуживания FIFO (первым пришел – первым ушел). В программировании очереди применяются при моделировании, диспетчеризации задач операционной системой, буферизованном вводе/выводе.

Графически ее удобно изобразить в виде вертикального списка (рис. 2), например, очередь в магазине, где для того чтобы дойти до кассы и не нарушить установленный порядок, нужно дождаться, пока все покупатели перед вами не приобретут товар. Разумеется, будут появляться новые покупатели, которые будут занимать свое место в очереди в ожидании покупки.

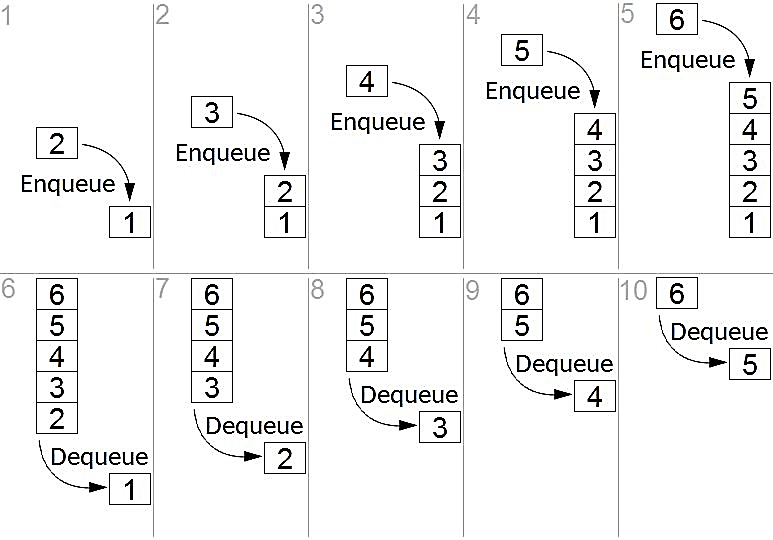


Рис. 2. Графическое представление очереди

Основные операции над очередями:

1. чтение первого элемента;
2. добавление нового элемента;
3. удаление существующего элемента.

Если для стека в момент добавления или удаления элемента допустимо задействование лишь его вершины, то касательно очереди эти две операции должны быть применены так, как это регламентировано в определении этой структуры данных, т. е. добавление – в конец, удаление – из начала.

Выделяют два способа программной реализации очереди. Первый основан на базе массива, второй – на базе указателей (связного списка). Первый способ – статический, так как очередь представляется в виде простого статического массива, второй – динамический.

Кольцевой буфер так же известен, как кольцевая очередь или циклический буфер и является распространенной формой очереди. Это популярный, легко реализуемый стандарт, и хотя он представлен в виде круга (рис. 3) в базовом коде он является линейным. Кольцевая очередь существует как массив фиксированной длины с двумя указателями: один представляет начало очереди, другой – хвост.

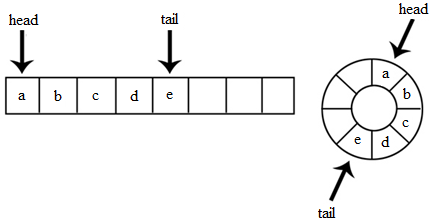


Рис. 3. Кольцевая очередь

Операции с очередями работают следующим образом.

1. Два указателя, называемые и , используются для отслеживания первого и последнего элементов в очереди.
2. При инициализации очереди значения и устанавливаются равными –1.
3. При добавлении элемента постепенно увеличивается значение индекса и помещается новый элемент в положение, на которое указывает .
4. При снятии очереди с элемента возвращается значение, на которое указывает , и постепенно увеличивается индекс .
5. Перед постановкой в очередь проверяется, заполнена ли очередь.
6. Перед снятием очереди проверяется, пуста ли очередь.
7. При инициализации первого элемента устанавливается значение в 0.
8. При удалении последнего элемента сбрасываются значения и в –1.

Недостатком метода является его фиксированный размер. Для очередей, где элементы должны быть добавлены и удалены в середине, а не только в начале и конце буфера, реализация в виде связанного списка является предпочтительным подходом.

Обратная польская запись (нотация) (рис. 4) – форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. Обратная польская запись имеет ряд преимуществ перед инфиксной записью при выражении алгебраических формул, одно из них то, что инфиксные операторы имеют приоритеты, которые произвольны и нежелательны.

|  |  |  |
| --- | --- | --- |
| Простое выражение | Прямая польская запись | Обратная польская запись |
| X + 3 \* Y  (X + 3) \* Y  1 + 2 | + X \* 3 Y  \* + X 3 Y  + 1 2 | X 3 Y \* +  X 3 + Y \*  1 2 + |

Рис. 4. Представление прямой и обратной польской записи

Обратная польская запись отлично подходит для вычисления выражений при помощи стека. Причем сам алгоритм достаточно прост. Необходимо просто прочитать обратную польскую запись слева направо. Если встречается операнд, то его нужно поместить в стек. Если встречается оператор, нужно выполнить заданную им операцию.

# **Постановка задачи**

Необходимо объединить лабораторные работы в единый проект. Нужно добавить инфраструктуру переключения между заданиями (интерактивное меню).

**Выполнение работы**

Для решения поставленных была создана программа на языке программирования C++. Итоговый код программы представлен в приложении А, а результат работы в приложении B.

Переключение между двумя практическими работами происходит при помощи инструкции множественного выбора.

# **Вывод**

В ходе проделанной работы были изучены структуры и их организация. Получены практические навыки работы со структурами. Структура – это совокупность переменных, объединенных одним именем, предоставляющая общепринятый способ совместного хранения информации. Объявление структуры приводит к образованию шаблона, используемого для создания объектов структуры. Стоит отметить, что структуры относятся к типам значений, и поэтому ими можно оперировать непосредственно, а не по ссылке. Следовательно, для работы со структурой вообще не требуется переменная ссылочного типа, а это означает в ряде случаев существенную экономию оперативной памяти. Также была проведена работа с динамическими массивами и списками: сравнение скорости вставки, удаление и получения элемента двусвязного списка и динамического массива. В последней практической получили практический навык работы со стеками и очередями; изучили обратную и прямую польскую нотацию; провели сравнительный анализ этих структур данных.

# **ПРИЛОЖЕНИЕ А**

**ПОЛНЫЙ КОД ПРОГРАММЫ**

#include <vector>

#include <iostream>

#include <fstream>

#include <string>

#include <chrono>

#include <cctype>

#include <stdlib.h>

#include <cstdio>

#include <stack>

using namespace std;

using namespace std::chrono;

struct Profile

{

string full\_name;

string sex;

int group;

int id;

int grades[8];

char depart[9];

string date;

};

//Данные о студентах с id i

void show(vector<Profile> student, int n, int i)

{

cout << "Full name: " << student[i].full\_name << "\n";

cout << "Group: " << student[i].group << "\n";

cout << "Index: " << student[i].id << "\n";

cout << "Sex (F/M): " << student[i].sex << "\n";

cout << "Department (day/evening/distance): " << student[i].depart << "\n";

cout << "Term grades: ";

for (int j = 0; j < 8; j++) cout << student[i].grades[j] << " "; cout << "\n";

cout << "Date and time of recording:" << student[i].date << "\n\n";

}

//Вывод всех данных о студентах

void showStud(vector<Profile> student, int n)

{

int stud;

cout << "All students on the list\n\n";

for (int i = 0; i < n; i++)

{

stud = i + 1;

cout << "Student " << stud << "\n\n";

cout << "Full name: " << student[i].full\_name << "\n";

cout << "Group: " << student[i].group << "\n";

cout << "Index: " << student[i].id << "\n";

cout << "Sex (F/M): " << student[i].sex << "\n";

cout << "Department (day/evening/distance): " << student[i].depart << "\n";

cout << "Term grades: ";

for (int j = 0; j < 8; j++) cout << student[i].grades[j] << " "; cout << "\n";

cout << "Date and time of recording:" << student[i].date << "\n\n";

}

cout << "\n\n";

system("pause");

system("cls");

}

//Создание новой записи о студенте

void add(vector<Profile>& student, int& n)

{

n++;

student.resize(n);

int i = n - 1;

ofstream fi;

fi.open("students.txt", fstream::app);

if (!fi.is\_open())

{

cout << "\nFile opening error";

}

else {

cout << "Enter full name: ";

cin.ignore();

getline(cin, student[i].full\_name);

fi << student[i].full\_name << "\n";

cout << "Enter group: ";

cin >> student[i].group;

fi << student[i].group << "\n";

cout << "Enter the id: ";

cin >> student[i].id;

fi << student[i].id << "\n";

cout << "Enter the gender of the person(F/M): ";

cin.ignore();

getline(cin, student[i].sex);

fi << student[i].sex << "\n";

cout << "Enter the form of study(day/evening/distance): ";

cin >> student[i].depart;

fi << student[i].depart << "\n";

for (int j = 0; j < 8; j++) { cout << "Enter grades: "; cin >> student[i].grades[j]; }

for (int j = 0; j < 8; j++) fi << student[i].grades[j] << " ";

fi << "\n";

cout << "Enter the date and time of the recording: ";

cin.ignore();

getline(cin, student[i].date);

fi << student[i].date << "\n";

}

fi.close();

cout << "\n\nRecorded\n\n";

system("pause");

system("cls");

}

//Запись нового студента в файл

void recording(vector<Profile>& student, int& n)

{

ofstream Rec;

Rec.open("students.txt");

if (!Rec.is\_open())

{

cout << "\nFile opening error";

}

else {

for (int i = 0; i < n; i++)

{

Rec << student[i].full\_name << "\n";

Rec << student[i].group << "\n";

Rec << student[i].id << "\n";

Rec << student[i].sex << "\n";

Rec << student[i].depart << "\n";

for (int j = 0; j < 8; j++) Rec << student[i].grades[j] << " ";

Rec << "\n";

Rec << student[i].date << "\n";

}

}

Rec.close();

}

//Внесение изменений в уже имеющуюся запись

void change(vector<Profile>& student, int& n)

{

int num, choose;

cout << "Enter student number from 1 to " << n << "\n";

cout << "You entered: ";

cin >> num;

num--;

cout << "\nWhat needs to be changed\n\n";

cout << "1 - full name\n";

cout << "2 - group number\n";

cout << "3 - id\n";

cout << "4 - sex\n";

cout << "5 - department (day/evening/distance)\n";

cout << "6 - grades\n";

cout << "7 - date and time of the recording\n";

cout << "\nYou entered: ";

cin >> choose;

cout << "\n";

switch (choose)

{

case 1:

cout << "Enter full name: "; cin.ignore();

getline(cin, student[num].full\_name);

break;

case 2:

cout << "Enter group: ";

cin >> student[num].group;

break;

case 3:

cout << "Enter the id: ";

cin >> student[num].id;

break;

case 4:

cout << "Enter the gender of the person(F/M): "; cin.ignore();

getline(cin, student[num].sex);

break;

case 5:

cout << "Enter the form of study(day/evening/distance): "; cin.ignore();

cin >> student[num].depart;

break;

case 6:

for (int j = 0; j < 8; j++) { cout << "Enter grades: "; cin >> student[num].grades[j]; }

break;

case 7:

cout << "Enter the date and time of the recording: "; cin.ignore();

getline(cin, student[num].date);

break;

}

recording(student, n);

cout << "\nData changed\n\n";

system("pause");

system("cls");

}

//Вывод информации обо всех студентах группы N

void showGroup(vector<Profile> student, int n)

{

int num, check = 0;

cout << "Enter group number";

cout << "\nYou entered: ";

cin >> num; cout << "\n";

for (int i = 0; i < n; i++)

{

if (student[i].group == num)

{

show(student, n, i);

check = 1;

}

}

if (check == 0) cout << "There is no such group";

cout << "\n\n";

system("pause");

system("cls");

}

//Вывод топа самых успешных студентов с наивысшим по рейтингу средним баллом за прошедшую сессию

void rating(vector<Profile> student, int n)

{

double\* rating = new double[n];

double max1 = 0, max2 = 0, max3 = 0, dub = 0.0;

int top1 = 0, top2 = 0, top3 = 0;

for (int i = 0; i < n; i++)

{

for (int j = 0; j < 8; j++)dub += student[i].grades[j];

dub /= 8;

rating[i] = dub;

dub = 0;

}

for (int i = 0; i < n; i++) { if (rating[i] > max1) { max1 = rating[i]; top1 = i; } }

for (int i = 0; i < n; i++) { if (rating[i] > max2 && student[i].full\_name != student[top1].full\_name) { max2 = rating[i]; top2 = i; } }

for (int i = 0; i < n; i++) { if (rating[i] > max3 && student[i].full\_name != student[top1].full\_name && student[i].full\_name != student[top2].full\_name) { max3 = rating[i]; top3 = i; } }

cout << "Top 1 with an average score: " << max1 << "\n\n"; show(student, n, top1);

cout << "\n\nTop 2 with an average score: " << max2 << "\n\n"; show(student, n, top2);

cout << "\n\nTop 3 with an average score: " << max3 << "\n\n"; show(student, n, top3);

delete[] rating;

cout << "\n\n";

system("pause");

system("cls");

}

//Вывод количества студентов мужского и женского пола

void gender(vector<Profile> student, int n)

{

int countM = 0, countF = 0;

for (int i = 0; i < n; i++)

{

if (student[i].sex == "F") countF++;

if (student[i].sex == "M") countM++;

}

cout << "\n\nWomen: " << countF << "\n\n";

cout << "Men: " << countM;

cout << "\n\n\n";

system("pause");

system("cls");

}

//Определение количества студентов, которые будут получать стипендию

void scholarship(vector<Profile> student, int n)

{

int check = 0, count = 0;

for (int i = 0; i < n; i++)

{

for (int j = 0; j < 8; j++) {

if ((student[i].grades[j] == 3) || (student[i].grades[j] == 2)) check = 1;

}

string a = student[i].depart;

if (a == "day" && check == 0) count++;

check = 0;

}

cout << "\n\nNumber of students with a scholarship: " << count << "\n\n\n";

system("pause");

system("cls");

}

//Вывод данных о студентах, которые не получают стипендию; учатся только на «хорошо» и «отлично»; учатся только на «отлично»

void noScholarship(vector<Profile> student, int n)

{

int check = 0, count = 0, count5 = 0, count5\_ = 0, count4 = 0;

cout << "Students who do not receive a scholarship:\n\n";

for (int i = 0; i < n; i++)

{

for (int j = 0; j < 8; j++)

if (student[i].grades[j] == 3 || student[i].grades[j] == 2) check = 1;

string a = student[i].depart;

if (a == "evening" || check == 1)

{

show(student, n, i);

}

check = 0;

}

system("pause"); system("cls");

cout << "Students with grades 5 and 4:\n\n";

for (int i = 0; i < n; i++)

{

for (int j = 0; j < 8; j++)

{

if (student[i].grades[j] == 2) check = 1;

if (student[i].grades[j] == 3) check = 1;

if (student[i].grades[j] == 5) count5\_++;

if (student[i].grades[j] == 4) count4++;

}

if (check == 0 && count5\_ < 8 && count4 > 0)

{

show(student, n, i);

}

count5\_ = 0;

check = 0;

}

system("pause"); system("cls");

cout << "Excellent students:\n\n";

for (int i = 0; i < n; i++)

{

for (int j = 0; j < 8; j++)

if (student[i].grades[j] == 5) count5++;

if (count5 == 8)

{

show(student, n, i);

}

count5 = 0;

}

cout << "\n\n";

system("pause");

system("cls");

}

//Вывод данных о студентах, имеющих номер в списке

void showID(vector<Profile> student, int n)

{

int num, check = 0;

cout << "Enter student's id\n";

cout << "You entered:";

cin >> num; cout << "\n";

for (int i = 0; i < n; i++)

{

if (student[i].id == num)

{

show(student, n, i);

check = 1;

}

}

if (check == 0) cout << "There is no such id";

cout << "\n\n";

system("pause");

system("cls");

}

//Вывод всех записей, сделанных в день, который введет пользователь

void time(vector<Profile> student, int n)

{

string t[12] = { "12","13","14","15","16","17","18","19","20","21","22","23" };

string t\_[12] = { "00","01","02","03","04","05","06","07","08","09","10","11" };

string sub, dub, time\_; int p, check = 0;

cout << "Enter day of week,month and day as in the file: ";

cin.ignore();

getline(cin, time\_); cout << "\n\n";

for (int i = 0; i < n; i++)

{

sub = student[i].date;

if (sub.find\_first\_not\_of(time\_) >= 10)

{

show(student, n, i);

check = 1;

}

}

if (check == 0) cout << "No notes made on this day\n\n";

system("pause"); system("cls");

cout << "Afternoon notes\n\n";

for (int i = 0; i < n; i++)

{

dub = student[i].date;

if (dub.size() == 24) p = 11;

else p = 10;

for (int j = 0; j < 12; j++)

{

if (t[j] == dub.substr(p, 2))

{

show(student, n, i);

}

}

}

system("pause"); system("cls");

cout << "Notes made before noon\n\n";

for (int i = 0; i < n; i++)

{

dub = student[i].date;

if (dub.size() >= 24) p = 11;

else p = 10;

for (int j = 0; j < 12; j++)

{

if (t\_[j] == dub.substr(p, 2))

{

show(student, n, i);

}

}

}

cout << "\n\n";

system("pause");

system("cls");

}

void PW1() {

ifstream fin;

fin.open("students.txt");

int n = 0, task, task2, task3;

while (!fin.eof()) {

if (fin.get() == '\n')

n++;

}

n = n / 7;

fin.close();

cout << "Number of students on the list: " << n << "\n\n";

vector<Profile>student(n);

ifstream f;

f.open("students.txt");

if (!f.is\_open())

{

cout << "\nFile opening error";

}

else {

for (int i = 0; i < n; i++)

{

getline(f, student[i].full\_name);

f >> student[i].group; f.ignore();

f >> student[i].id; f.ignore();

getline(f, student[i].sex);

f >> student[i].depart; f.ignore();

for (int j = 0; j < 8; j++) { f >> student[i].grades[j]; }

f.ignore(2);

getline(f, student[i].date);

}

}

f.close();

do

{

cout << "Enter number of task\n\n";

cout << "1. Create a new student record\n";

cout << "2. Making changes to an existing record\n";

cout << "3. Display of all student data\n";

cout << "4. Display of information about all students of group N\n";

cout << "5. Finding the most successful students\n";

cout << "6. Display of the number of male and female students\n";

cout << "7. Determining the number of students who will receive the scholarship\n";

cout << "8. Conclusion of data on students who: do not receive a scholarship, study only for good and excellent, study only excellently\n";

cout << "9. Displays data about students who have a number in the list - k\n";

cout << "10. Displays all entries made on the day that the user enters, displays all entries made in the afternoon, displays all entries made before noon.\n";

cout << "To exit, enter 0\n";

cout << "\n";

cout << "You entered: ";

cin >> task;

system("cls");

switch (task)

{

case 1:

add(student, n);

break;

case 2:

change(student, n);

break;

case 3:

showStud(student, n);

break;

case 4:

showGroup(student, n);

break;

case 5:

rating(student, n);

break;

case 6:

gender(student, n);

break;

case 7:

scholarship(student, n);

break;

case 8:

noScholarship(student, n);

break;

case 9:

showID(student, n);

break;

case 10:

time(student, n);

break;

case 0:

break;

}

} while (task != 0);

}

struct list

{

int data;

list\* prev;

list\* next;

};

int count\_ = 0;

list\* head = NULL, \* tail = NULL;

//поиск индекса элемента num

int SerchInd\_l(int num)

{

list\* t = head;

int serch\_c = 0;

while (t) {

if (t->data == num)

{

return serch\_c;

break;

}

t = t->next;

serch\_c++;

}

}

//поиск элемента с индексом num

list\* SerchData\_l(int num)

{

list\* t = head;

int serch\_c = 0;

while (t) {

if (serch\_c == num)

{

return t;

break;

}

t = t->next;

serch\_c++;

}

}

int SerchData\_l3(int num)

{

list\* t = head;

int serch\_c = 0;

while (t) {

if (serch\_c == num)

{

return t->data;

break;

}

t = t->next;

serch\_c++;

}

}

//заполнение списка

void Add\_list(int x) {

list\* temp = new list;

temp->data = x;

temp->next = NULL;

count\_++;

if (!head) {

temp->prev = NULL;

head = temp;

tail = head;

}

else {

temp->prev = tail;

tail->next = temp;

tail = temp;

}

}

//вставка элемента не в начало списка

void Add\_In(list\* elem, int num)

{

list\* NewEl = new list;

NewEl->data = num;

if (elem->next == NULL) NewEl->next = NULL;

else NewEl->next = elem->next;

elem->next = NewEl;

NewEl->prev = elem;

if (elem->next != NULL) elem->next->prev = NewEl;

if (NewEl->next == NULL) tail = NewEl;

}

//вставка элемента в начало списка

void First\_In(int num)

{

list\* NewEl = new list;

list\* tmp = head;

NewEl->prev = NULL;

NewEl->next = head;

NewEl->data = num;

tmp->prev = NewEl;

tmp->next = head->next;

head = NewEl;

}

//вставка элемента

void Insert\_list(int in, int ch)

{

count\_++;

int ind, num = ch, c = 0;

ind = in;

list\* per = head;

while (per != NULL)

{

if (ind == 0)

{

First\_In(num);

break;

}

if (c == ind - 1)

{

Add\_In(per, num);

break;

}

per = per->next; c++;

}

}

//вывод списка

void Show\_list() {

list\* t = head;

while (t) {

cout << t->data << " ";

t = t->next;

}

cout << "\n\n";

}

//удаление элемента по индексу

void Del\_list(int x)

{

if ((x == 0) and (head->next)) {

list\* temp = head;

head = head->next;

head->prev = NULL;

delete temp;

count\_--;

return;

}

else if ((x == 0) and (head == tail)) {

head->next = NULL;

head = NULL;

delete head;

count\_ = 0;

return;

}

if (x == count\_ - 1) {

list\* temp = tail;

tail = tail->prev;

tail->next = NULL;

delete temp;

count\_--;

return;

}

list\* temp = SerchData\_l(x);

list\* prev\_el, \* next\_el;

temp->prev->next = temp->next;

temp->next->prev = temp->prev;

delete temp;

count\_--;

}

//заполнение списка 1 способом

void Fill\_list1(int N)

{

int elem;

for (int i = 0; i < N; i++)

{

elem = rand() % 199 - 99;

Add\_list(elem);

}

}

//заполнение списка 2 способом

void Fill\_list3()

{

ifstream f;

f.open("file.txt");

if (!f.is\_open())

{

cout << "\nFile opening error";

}

else {

int i = 0, x;

while (!f.eof())

{

f >> x;

Add\_list(x);

i++;

}

}

f.close();

}

//заполнение списка 3 способом

void Fill\_list2()

{

int tmp = 0;

while (tmp != -1)

{

cin >> tmp;

if (tmp == -1)return;

Add\_list(tmp);

}

system("cls");

}

//вывод массива

void Show\_mas(int const\* mas\_dyn, int const& dyn)

{

for (int i = 0; i < dyn; i++)

{

cout << mas\_dyn[i] << " ";

}

}

//заполнение массива 1 способом

void Fill\_mas1(int& dyn\_a, int\* mas\_dyn)

{

for (int i = 0; i < dyn\_a; i++)

{

mas\_dyn[i] = rand() % 199 - 99;

}

}

//заполнение массива 2 способом

void Fill\_mas2(int& dyn\_b, int\* mas\_dyn)

{

int tmp = 0;

while (tmp != -1)

{

cin >> tmp;

if (tmp == -1)return;

mas\_dyn[dyn\_b] = tmp;

dyn\_b++;

}

system("cls");

}

//заполнение массива 3 способом

void Fill\_mas3(int& dyn\_c, int\* mas\_dyn)

{

ifstream f;

f.open("file.txt");

if (!f.is\_open())

{

cout << "\nFile opening error";

}

else {

int i = 0;

while (!f.eof())

{

f >> mas\_dyn[i];

i++;

}

dyn\_c = i;

}

f.close();

}

//расширение массива на 1

void Resize\_P(int& N, int\*& mas\_dyn)

{

int N2; N2 = N + 1;

int\* mas\_rez = new int[N2];

for (int i = 0; i < N2; i++)

mas\_rez[i] = mas\_dyn[i];

delete[] mas\_dyn;

mas\_dyn = mas\_rez;

}

//уменьшение массива на 1

void Resize\_M(int& N, int\*& mas\_dyn)

{

int N2; N2 = N - 1;

int\* mas\_rez = new int[N2];

for (int i = 0; i < N2; i++)

mas\_rez[i] = mas\_dyn[i];

delete[] mas\_dyn;

mas\_dyn = mas\_rez;

N = N2;

}

//вставка элемента в массив

void Insert\_mas(int& N, int\*& mas\_dyn, int ind, int ch)

{

int newi = ind; int i\_num = ch;

Resize\_P(N, mas\_dyn);

for (int i = N - 1; i >= newi; i--)

mas\_dyn[i + 1] = mas\_dyn[i];

mas\_dyn[newi] = i\_num;

N++;

}

//удаление элемента по индексу

void Del\_mas(int& N, int\*& mas\_dyn, int del)

{

for (int i = del; i < N - 1; i++)

mas\_dyn[i] = mas\_dyn[i + 1];

Resize\_M(N, mas\_dyn);

}

//поиск индекса ind элемента ch

int SerchInd\_m(int N, int\* dyn\_mas, int ch)

{

for (int i = 0; i < N; i++)

{

if (dyn\_mas[i] == ch) return i;

}

}

//поиск элемента с индексом ind

int SerchData\_m(int N, int\* dyn\_mas, int ind)

{

for (int i = 0; i < N; i++)

{

if (i == ind) return dyn\_mas[i];

}

}

void Task\_fill\_time1(int\*& mas\_dyn, int& N)

{

auto begin\_m = steady\_clock::now();

Fill\_mas1(N, mas\_dyn);

auto end\_m = steady\_clock::now();

auto elepsed\_m = duration\_cast<nanoseconds>(end\_m - begin\_m);

Show\_mas(mas\_dyn, N);

cout << "\n\nTime spent in nanoseconds: " << elepsed\_m.count() << "\n\n\n";

auto begin\_l = steady\_clock::now();

Fill\_list1(N);

auto end\_l = steady\_clock::now();

auto elepsed\_l = duration\_cast<nanoseconds>(end\_l - begin\_l);

Show\_list();

cout << "Time spent in nanoseconds: " << elepsed\_l.count() << "\n";

}

void Task\_fill\_time2(int\*& mas\_dyn, int& N)

{

auto fillmasB = steady\_clock::now();

Fill\_mas2(N, mas\_dyn);

auto fillmasE = steady\_clock::now();

auto elepsed\_mas = duration\_cast<nanoseconds>(fillmasE - fillmasB);

Show\_mas(mas\_dyn, N);

cout << "\n\nTime spent in nanoseconds: " << elepsed\_mas.count() << "\n\n\n";

auto filllistB = steady\_clock::now();

Fill\_list2();

auto filllistE = steady\_clock::now();

auto elepsed\_list = duration\_cast<nanoseconds>(filllistE - filllistB);

Show\_list();

cout << "Time spent in nanoseconds: " << elepsed\_list.count() << "\n";

}

void Task\_fill\_time3(int\*& mas\_dyn, int& N)

{

auto begin\_m = steady\_clock::now();

Fill\_mas3(N, mas\_dyn);

auto end\_m = steady\_clock::now();

auto elepsed\_m = duration\_cast<nanoseconds>(end\_m - begin\_m);

Show\_mas(mas\_dyn, N);

cout << "\n\nTime spent in nanoseconds: " << elepsed\_m.count() << "\n\n\n";

auto begin\_l = steady\_clock::now();

Fill\_list3();

auto end\_l = steady\_clock::now();

auto elepsed\_l = duration\_cast<nanoseconds>(end\_l - begin\_l);

Show\_list();

cout << "Time spent in nanoseconds: " << elepsed\_l.count() << "\n";

}

void Task\_insert\_time(int\*& mas\_dyn, int& N)

{

int ind, ch;

cout << "Array\n"; Show\_mas(mas\_dyn, N);

cout << "\n\nEnter index to insert: "; cin >> ind;

cout << "Enter new element: "; cin >> ch;

auto begin\_m = steady\_clock::now();

Insert\_mas(N, mas\_dyn, ind, ch);

auto end\_m = steady\_clock::now();

auto elepsed\_m = duration\_cast<nanoseconds>(end\_m - begin\_m);

cout << "\nArray after insertion\n"; Show\_mas(mas\_dyn, N);

cout << "\n\nTime spent in nanoseconds: " << elepsed\_m.count() << "\n\n\n";

cout << "List\n"; Show\_list();

cout << "Enter index to insert: "; cin >> ind;

cout << "Enter new element: "; cin >> ch; cout << "\n";

auto begin\_l = steady\_clock::now();

Insert\_list(ind, ch);

auto end\_l = steady\_clock::now();

auto elepsed\_l = duration\_cast<nanoseconds>(end\_l - begin\_l);

cout << "\List after insertion\n"; Show\_list();

cout << "Time spent in nanoseconds: " << elepsed\_l.count() << "\n\n";

}

void Task\_del\_time(int\*& mas\_dyn, int& N)

{

int ch = 0, e, pr;

cout << "if insert in array by value enter 1 if at index enter 2\n";

cout << "You choose: "; cin >> pr; cout << "\n\n" << "Array\n";

Show\_mas(mas\_dyn, N); cout << "\n\n";

if (pr == 1) { cout << "Enter the item to delete: "; cin >> e; ch = SerchInd\_m(N, mas\_dyn, e); cout << "\n"; }

if (pr == 2) { cout << "Enter index to delete: "; cin >> ch; cout << "\n"; }

auto begin\_m = steady\_clock::now();

Del\_mas(N, mas\_dyn, ch);

auto end\_m = steady\_clock::now();

auto elepsed\_m = duration\_cast<nanoseconds>(end\_m - begin\_m);

cout << "Array after element has been removed\n"; Show\_mas(mas\_dyn, N);

cout << "\n\nTime spent in nanoseconds: " << elepsed\_m.count() << "\n\n\n";

cout << "\nif insert in list by value enter 1 if at index enter 2\n";

cout << "You choose: "; cin >> pr; cout << "\n" << "list\n";

Show\_list();

if (pr == 1) { cout << "Enter the item to delete: "; cin >> e; ch = SerchInd\_l(e); cout << "\n"; }

if (pr == 2) { cout << "Enter index to delete: "; cin >> ch; cout << "\n"; }

auto begin\_l = steady\_clock::now();

Del\_list(ch);

auto end\_l = steady\_clock::now();

auto elepsed\_l = duration\_cast<nanoseconds>(end\_l - begin\_l);

cout << "List after element has been removed\n"; Show\_list();

cout << "Time spent in nanoseconds: " << elepsed\_l.count() << "\n\n";

}

void Add(int\*& mas\_dyn, int& N)

{

int ch = 0, e, pr;

cout << "\n" << "Array\n";

Show\_mas(mas\_dyn, N); cout << "\n\n";

auto begin\_m = steady\_clock::now();

for (int i = 0; i < N; i++) {

Del\_mas(N, mas\_dyn, i);

}

auto end\_m = steady\_clock::now();

auto elepsed\_m = duration\_cast<nanoseconds>(end\_m - begin\_m);

cout << "Array after element has been removed\n"; Show\_mas(mas\_dyn, N);

cout << "\n\nTime spent in nanoseconds: " << elepsed\_m.count() << "\n\n\n";

cout << "\n" << "list\n";

Show\_list();

auto begin\_l = steady\_clock::now();

for (int i = 0; i < N; i++) {

Del\_list(i);

}

auto end\_l = steady\_clock::now();

auto elepsed\_l = duration\_cast<nanoseconds>(end\_l - begin\_l);

cout << "List after element has been removed\n"; Show\_list();

cout << "Time spent in nanoseconds: " << elepsed\_l.count() << "\n\n";

}

void Task\_find\_time(int\*& mas\_dyn, int& N)

{

int Find\_m, Fch\_m, Find\_l, Fch\_l, choose\_;

cout << "If you need to get the index of the entered number, press 1,if the number with a specific index, press 2\n";

cout << "You choose: "; cin >> choose\_;

//получение числа

if (choose\_ == 1)

{

cout << "\n\nArray\n";

Show\_mas(mas\_dyn, N); cout << "\n\n";

cout << "\nEnter number: "; cin >> Fch\_m;

cout << "Number index: ";

auto begin\_m = steady\_clock::now();

cout << SerchInd\_m(N, mas\_dyn, Fch\_m);

auto end\_m = steady\_clock::now();

auto elepsed\_m = duration\_cast<nanoseconds>(end\_m - begin\_m);

cout << "\n\nTime spent in nanoseconds: " << elepsed\_m.count();

cout << "\n\n\nlist\n";

Show\_list(); cout << "\n";

cout << "Enter number: "; cin >> Fch\_l;

cout << "Number with index: ";

auto begin\_l = steady\_clock::now();

cout << SerchInd\_l(Fch\_l);

auto end\_l = steady\_clock::now();

auto elepsed\_l = duration\_cast<nanoseconds>(end\_l - begin\_l);

cout << "\n\nTime spent in nanoseconds: " << elepsed\_l.count() << "\n\n";

}

//получение индекса

if (choose\_ == 2)

{

cout << "\n\nArray\n";

Show\_mas(mas\_dyn, N); cout << "\n\n";

cout << "\nEnter index: "; cin >> Find\_m;

auto begin\_m = steady\_clock::now();

cout << "Number: ";

cout << SerchData\_m(N, mas\_dyn, Find\_m);

auto end\_m = steady\_clock::now();

auto elepsed\_m = duration\_cast<nanoseconds>(end\_m - begin\_m);

cout << "\n\nTime spent in nanoseconds: " << elepsed\_m.count() << "\n\n";

cout << "\n\n\nlist\n";

Show\_list(); cout << "\n";

cout << "Enter index: "; cin >> Find\_l;

cout << "Number: ";

auto begin\_l = steady\_clock::now();

cout << SerchData\_l3(Find\_l);

auto end\_l = steady\_clock::now();

auto elepsed\_l = duration\_cast<nanoseconds>(end\_l - begin\_l);

cout << "\n\nTime spent in nanoseconds: " << elepsed\_l.count() << "\n\n";

}

}

void PW2()

{

int N = 100, fill, task;

int\* mas\_dyn = new int[N];

cout << "Forming an integer one-dimensional array and a doubly connected list of dimension N, where: \n\n";

cout << "1) the user enters the number of elements in the array, which will be automatically filled with random numbers (0 to 99);\n";

cout << "2) the user enters the array elements into the console, N is determined automatically by the number of elements entered;\n\n";

cout << "3) \* an array is read from a file, N is defined as the number of array elements in the file.\n\n";

cout << "You entered: ";

cin >> fill; cout << "\n\n";

system("pause"); system("cls");

switch (fill)

{

case 1:

cout << "Enter the number of items\n" << "You entered: ";

cin >> N; cout << "\n\n";

Task\_fill\_time1(mas\_dyn, N);

break;

case 2:

N = 0;

Task\_fill\_time2(mas\_dyn, N);

break;

case 3:

Task\_fill\_time3(mas\_dyn, N);

break;

}

cout << "\n\n";

do

{

cout << "Choose a task:\n";

cout << "1) Comparison of insert time\n";

cout << "2) Comparison of delete time\n";

cout << "3) Comparison of time of receipt\n";

cout << "4) Removing even and odd elements\n";

cout << "Exit - 0\n\n";

cout << "You entered: "; cin >> task;

system("cls");

switch (task)

{

case 1:

Task\_insert\_time(mas\_dyn, N);

system("pause"); system("cls");

break;

case 2:

if (count\_ == 0 && N == 0) { cout << "List and array is empty"; system("pause"); system("cls"); break; }

if (count\_ == 0) { cout << "List is empty"; system("pause"); system("cls"); break; }

if (N == 0) { cout << "Array is empty"; system("pause"); system("cls"); break; }

Task\_del\_time(mas\_dyn, N);

system("pause"); system("cls");

break;

case 3:

Task\_find\_time(mas\_dyn, N);

system("pause"); system("cls");

break;

case 4:

if (count\_ == 0 && N == 0) { cout << "List and array is empty"; system("pause"); system("cls"); break; }

if (count\_ == 0) { cout << "List is empty"; system("pause"); system("cls"); break; }

if (N == 0) { cout << "Array is empty"; system("pause"); system("cls"); break; }

Add(mas\_dyn, N);

system("pause"); system("cls");

break;

case 0:

break;

}

} while (task != 0);

while (head) {

tail = head->next;

delete head;

head = tail;

}

delete[] mas\_dyn;

}

int priority(char o)

{

switch (o)

{

case '(': return 1;

case '+': case '-': return 2;

case '\*': case '/': return 3;

}

}

bool sign(char c)

{

return c == '+' || c == '-' || c == '\*' || c == '/';

}

bool num(char c)

{

return c >= '0' && c <= '9';

}

void RPN(char\* a, char\* out)

{

stack <char> S;

int i = 0, j = 0;

for (; a[i] != '\0'; ++i)

{

if (num(a[i]))

{

out[j] = a[i];

++j;

}

else

{

if (j != 0 && num(out[j - 1]))

{

out[j] = ' ';

++j;

}

if (sign(a[i]))

{

if (S.empty() || priority(S.top()) < priority(a[i]))

{

S.push(a[i]);

}

else

{

while (!S.empty() && (priority(S.top()) >= priority(a[i])))

{

out[j++] = S.top();

S.pop();

}

S.push(a[i]);

}

}

else

{

if (a[i] == '(')

{

S.push(a[i]);

}

else if (a[i] == ')')

{

if (S.empty() || S.top() == '(')

{

cout << "Input error!";

exit(1);

}

else

{

while (S.top() != '(')

{

out[j] = S.top();

S.pop();

j++;

}

}

S.pop();

}

else if (a[i] != ' ')

{

cout << "Initialize the variable " << a[i] << ": ";

cin >> a[i];

i--;

}

}

}

}

while (!S.empty())

{

if (S.top() == '(')

{

cout << "Input error!";

exit(1);

}

else

{

out[j] = S.top();

S.pop();

j++;

}

}

}

int calc(char\* out)

{

int j = 0, r1 = 0, r2 = 0;

stack <double> S;

stack <double> S1;

char\* pEnd = nullptr;

while (out[j] != '\0')

{

if (out[j] == '\_') {

++j;

continue;

}

if (out[j] >= '0' && out[j] <= '9')

{

long iNum = strtol(&out[j], &pEnd, 10);

S.push(iNum);

j += pEnd - &out[j];

}

else

{

if (sign(out[j]))

{

if (S.empty()) return false;

else {

r1 = S.top(); S.pop();

if (S.empty()) return false;

r2 = S.top(); S.pop();

switch (out[j])

{

case '+': S.push(r2 + r1); break;

case '-': S.push(r2 - r1); break;

case '\*': S.push(r2 \* r1); break;

case '/': S.push(r2 / r1); break;

}

}

}

++j;

}

}

return (S.top());

}

bool check(char\* a) {

bool right = true;

if (a[0] == '+' || a[0] == '\*' || a[0] == '/') {

right = false;

}

else if (a[strlen(a) - 1] == '+' || a[strlen(a) - 1] == '-' || a[strlen(a) - 1] == '\*' || a[strlen(a) - 1] == '/') {

right = false;

}

else {

for (int i = 0; i < strlen(a); i++) {

if ((a[i] == '+' || a[i] == '-' || a[i] == '\*' || a[i] == '/') && (a[i + 1] == '+' || a[i + 1] == '\*' || a[i + 1] == '/')) {

right = false;

}

if ((a[i] == '/') && (a[i + 1] == '0')) {

right = false;

}

}

}

return right;

}

void ex1() {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "1. Converting the entered expression: " << endl;

char a[100] = { 0 };

char out[100] = { 0 };

cout << "Expression: ";

gets\_s(a);

RPN(a, out);

cout << "RPN: " << out << endl << endl;

}

void ex2\_1() {

cout << endl << "2. 1)Checking for the correctness of a simple expression: " << endl;

char chek[100] = { 0 };

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Expression: ";

gets\_s(chek);

if (check(chek)) cout << "The expression is correct!" << endl;

else cout << "The expression is incorrect!" << endl;

}

void ex2\_2() {

cout << endl << " 2)Checking the RPN" << endl;

char chek[100] = { 0 };

//char cheked[100] = { 0 };

cout << "Expression: ";

gets\_s(chek);

//OPN(chek, cheked);

if (calc(chek)) cout << "The expression is correct!" << endl;

else cout << "The expression is incorrect!" << endl;

}

void ex3\_1() {

cout << endl << "3. 1)Evaluating a simple expression" << endl;

char a[100] = { 0 };

char out[100] = { 0 };

cout << "Expression: ";

gets\_s(a);

RPN(a, out);

cout << "Answer: " << calc(out) << endl << endl;

}

void ex3\_2() {

cout << endl << " 2)Evaluating a RPN" << endl;

char a[100] = { 0 };

cout << "Expression: ";

gets\_s(a);

cout << "Answer: " << calc(a) << endl << endl;

system("pause");

}

void PW3()

{

cout << "Expression input is implemented only from the keyboard" << endl << endl;

ex1();

ex2\_1();

ex2\_2();

ex3\_1();

ex3\_2();

system("cls");

}

int main()

{

int practicalWork = 1;

while (practicalWork != 0) {

system("CLS");

cout << "To choose Practical work, Enter the number of practical work or enter 0 for exit: \n";

cout << "\t1) Study and organization of structures \n";

cout << "\t2) Exploring the properties and organization of dynamic arrays and doubly linked lists \n";

cout << "\t3) Polish notation. Stack and queue \n";

cin >> practicalWork;

switch (practicalWork) {

case 1:

system("CLS");

PW1();

break;

case 2:

system("CLS");

PW2();

break;

case 3:

system("CLS");

PW3();

break;

case 0:

cout << "Have a nice day!\n";

break;

default:

cout << "Oh, no! This task doesn't exist. Let's try again.\n";

break;

}

}

return 0;

}

# **ПРИЛОЖЕНИЕ B**

**РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

**Интерактивное меню выбора практической работы:**

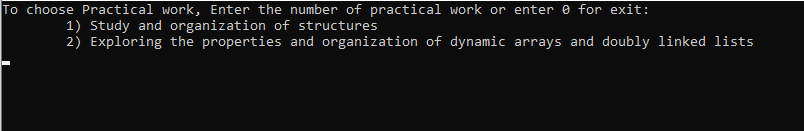
****

Рисунок 1 - интерактивное меню

**Первая практическая работа:**

Необходимо создать массив структур, содержащий информацию о студентах: ФИО, пол, номер группы, номер в списке группы, оценки за прошедшую сессию (всего 3 экзамена и 5 дифференцированных зачетов), форма обучения\*, отметка времени о внесении или изменении данных\*. Ввод и изменение данных обо всех студентах должен осуществляться в файл students\*:

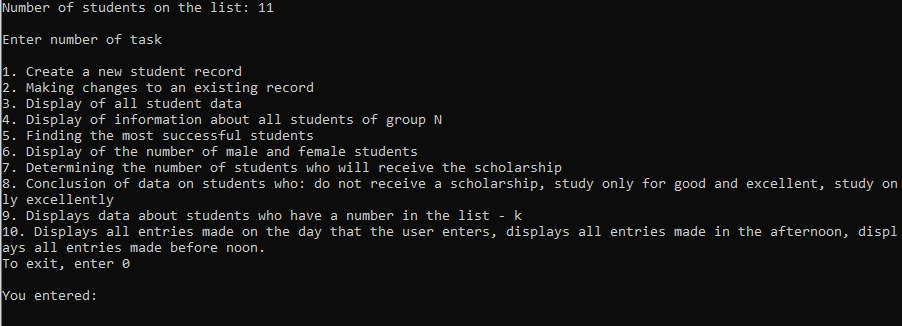


Рисунок 2 - Задание

1. Создание новой записи о студенте:

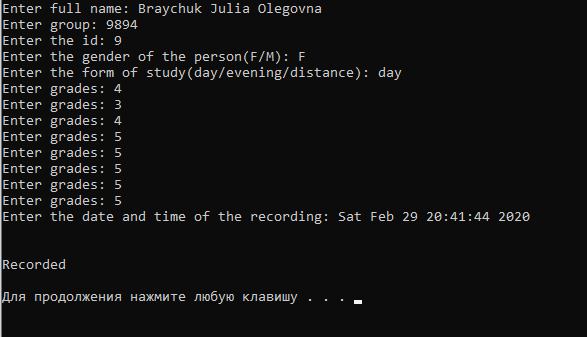


Рисунок 3 - Задание 1

1. Внесение изменений в уже имеющуюся запись:

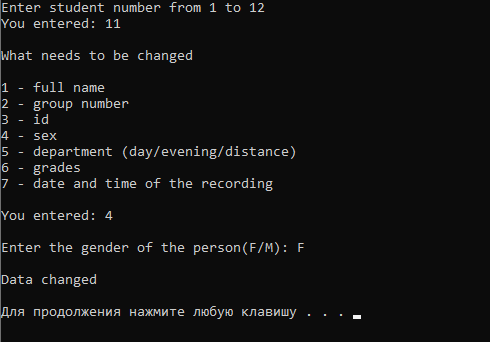


Рисунок 4 - Задание 2

1. Вывод всех данных о студентах:

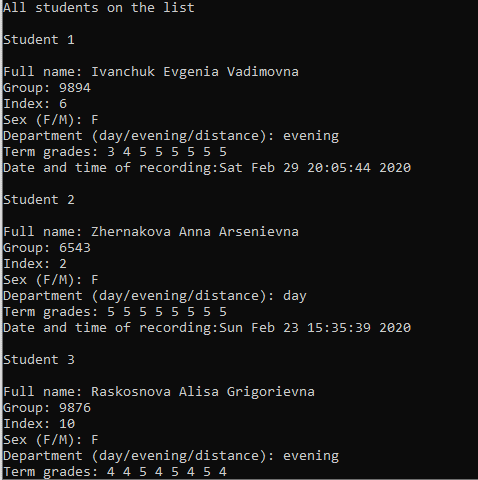
****

Рисунок 5 - Задание 3

1. Вывод информации обо всех студентах группы N. N – инициализируется пользователем:

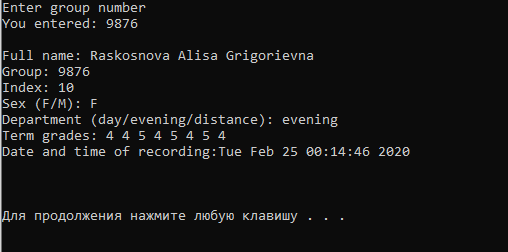


Рисунок 6 - Задание 4

1. Вывод топа самых успешных студентов с наивысшим по рейтингу средним баллом за прошедшую сессию:

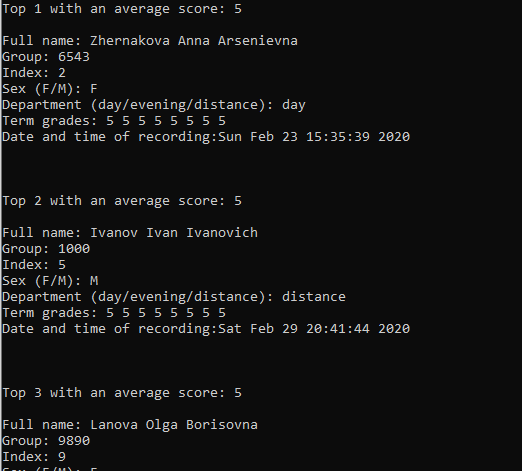


Рисунок 7 - Задание 5

1. Вывод количества студентов мужского и женского пола:

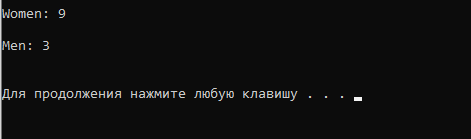
****

Рисунок 8 - Задание 6

1. Определение количества студентов, которые будут получать стипендию (стипендия начисляется, если у студента нет троек и очная форма обучения) :

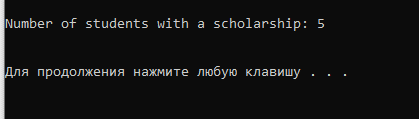


Рисунок 9 - Задание 7

1. Вывод данных о студентах, которые не получают стипендию; учатся только на «хорошо» и «отлично»; учатся только на «отлично» :

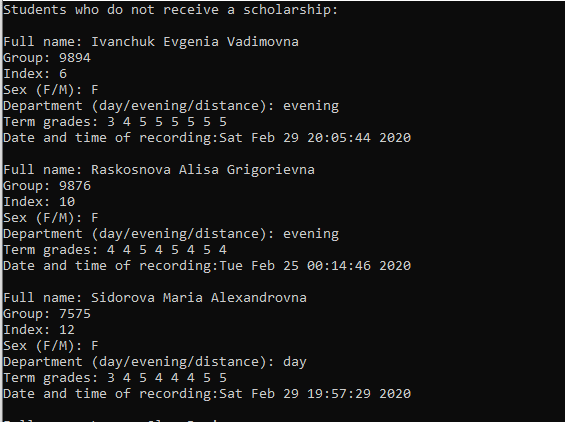
****

Рисунок 10 - Задание 8

1. Вывод данных о студентах, имеющих номер в списке – k:

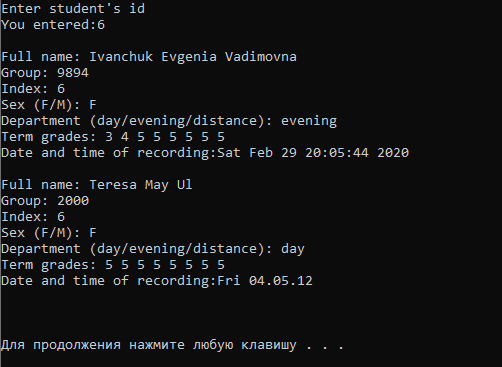


Рисунок 11 - Задание 9

1. Вывод всех записей, сделанных в день, который введет пользователь. Вывод всех записей, сделанных после полудня. Вывод всех записей, сделанных до полудня:

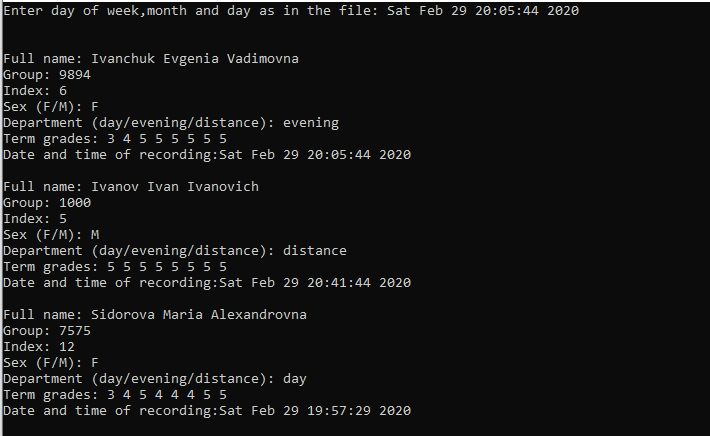


Рисунок 12 - Задание 10

**Вторая практическая работа:**

1. Формирование целочисленного одномерного массива размерности N, где:

a) пользователь вводит количество элементов в массиве, который будет автоматически заполняться случайными числами (0 до 99);

б) пользователь вводит в консоль элементы массива, N определяется автоматически по количеству введенных элементов;

в) \* массив считывается с файла, N определяется как количество элементов массива в файле.

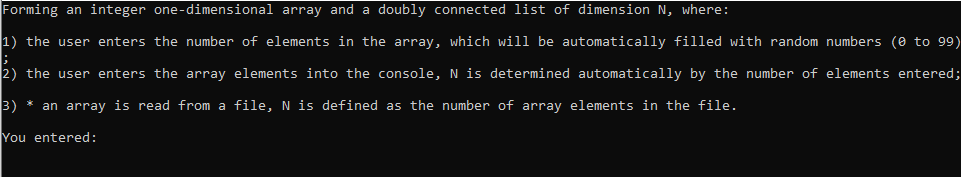


Рисунок 13 – Задание 1 и 5

1. Определение скорости создания динамического массива п. 1:

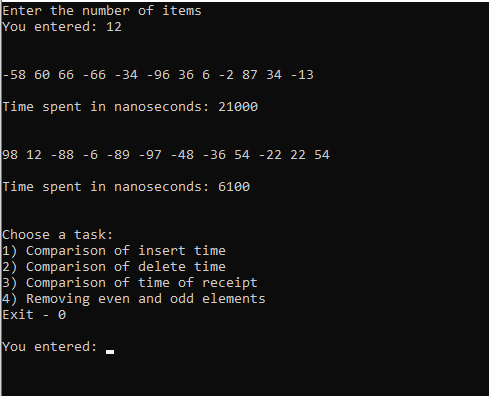


Рисунок 14 – Задание 2 и 6

1. Определение скорости создания динамического массива п. 1

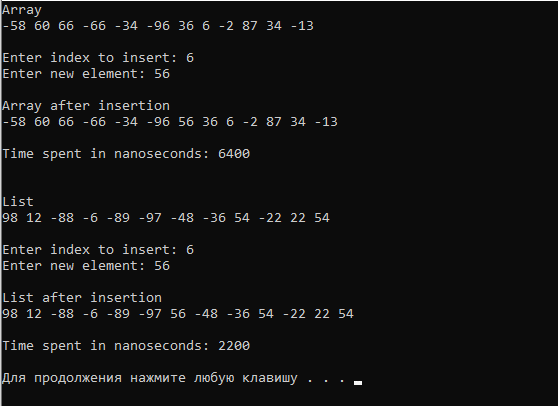


Рисунок 15 – Задание 3 и 7

1. Определение скорости вставки, удаления и получения элемента массива п. 3. :

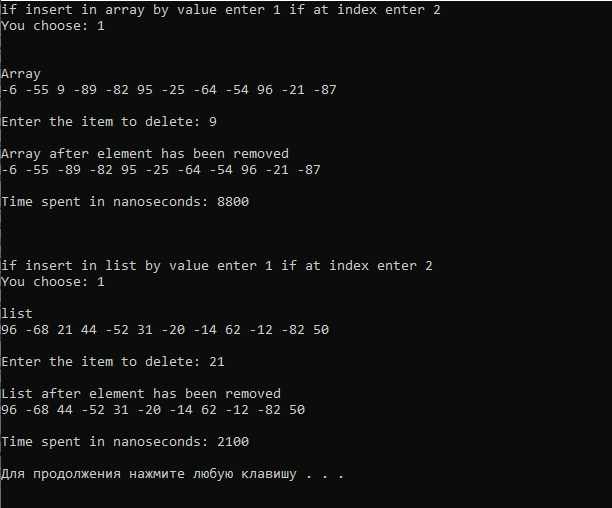


Рисунок 16 – Задание 4 и 8

**Третья практическая работа:**

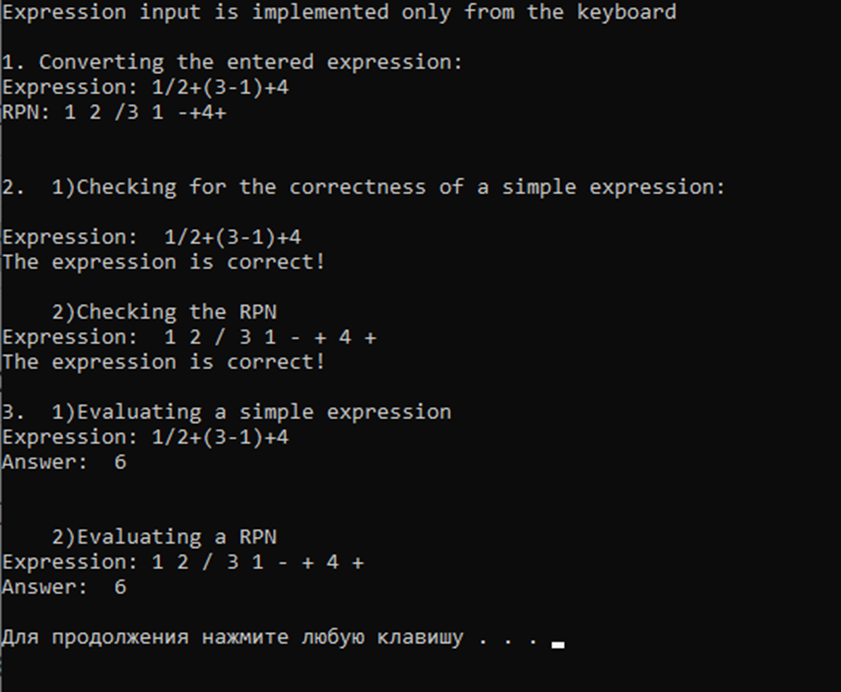


Рисунок 17 – Третья практическая